University: An_najah National University

Course: Distributed Operation Systems

Constructor: Dr.Samer Arandi.

Title: Turning the Bazar into an Amazon: Replication, Caching and Consistency.

Student's names: 1- Lamya Hanani. 2- Amani Zaqzouq.

Part1: Caching:

1- Cache class:

In this part we create a class in front service project to implement caching.

This class using Delay Queue (import java.util.concurrent.DelayQueue) which allows adding elements to queue with delay (TTL), so we can schedule removing of expired objects.

The cache_in_memry class consist of 3 basics functions:

1- add (key,value,period): adds return value of http request to cache temporarily for a period of time, and in out code we set this period to 3 minutes(180000 milisec).

2- remove (key): Removes the object which has the key(before the expiration time for this object ends).

3- get(key): Returns the object that has the key from the cache.

2-How we use caching in front server?

The caching process is essential for reading requests which they are in our case lookup and search requests.

1- lookup: when user sends lookup request the handler function first checks if this request in the cache, if yes it returns the result for the user, if no it sets up a connection with catalog server and sending request to

it, and after response returns, the handler sends it to the user and adds it to cache for three minutes (during this period if the user sends this request again the response returns from the cache).

The values that storing in the cache: key: id number for the book, value: book information, period: 3 minutes.

2- Search: The caching for this request is implemented at same way as the lookup caching is implemented.

Values in cache: Key: topic of the book, value: all topics belong to this topic, period: 3 minutes.

3- Buy: Although this request response doesn't store in the cache, but this request effects directly on the data stored in the cache.

If the user wants to buy the book that has id=1, then the amount of this book will decrease, so the information of this book will be out of date if it stored in the cache, so the buy handler function check if the information of this book in the cache, if yes it removes it from the cache, for the same purpose it checks also if the search response for the topic of this book stored in the cache, if yes it also removes it.

Part2: Replication:

In this part we made a copy of each catalog and order server, so now the Bazar website has five web services instead of three (two catalog, two order, one front).

Logically we should run each one of these services on separate device (or in our case on separate operation system) but our computers couldn't handle more than two virtual machines and the host operating system so:

We run each service and its copy on one os with the same ip but different ports:

Catalog1: 192.168.43.254:8080 (Runs on host os).

Catalog2: 192.168.43.254:8081. (Runs on host os).

Order1: 192.168.56.101:8001 (Runs on VM).

Order2: 192.168.56.101:8007 (Runs on VM).

Front: 192.168.56.102:8005 (Runs on VM).

Core idea: we used Round Robin algorithm concept to implement load balancing.

Lookup and search requests related to catalog service and buy related to order service, so to balance load on the catalog and order service from front service we do the following:

1- We defined a global variable called replic_value_l_s this for lookup and search requests, the first time lookup or search request is sent, this variable will be incremented by one and the request handler will connect with the catalog1 service, at the second request the handler will connect with catalog2 service and so on.
(If the variable value is odd then the handler will connect with catalog1 and if the variable value is even it will connect with catalog2).
** The lookup and search request handler increment global variable value.

2- We did as the above for buy request to balance the load on order service1 and order service two.

3-Because the order service sends requests to the catalog service (to check if the book exists in the database) we also did load balancing idea in the order service to balance load on catalog service when sending

lookup requests to the catalog, and we used the same principle in the implementation as in the front service.
In addition to that when the order service updates the information of the book we implemented it to update the database for each catalog service.

Experimental Evaluation and Measurements:

1-
Without caching:

Response time (ms)/request	search	lookup	buy
1	853	1024	667
2	45	789	232
3	125	107	243
4	55	309	126
5	42	199	183
6	41	111	222
7	39	21	65

Search Average response time = 1200/7= 171ms.

Lookup Average response time= 2560/7= 365 ms.

Buy Average response time = 1738/7= 248.28 ms.

With caching:

Response time (ms)/request	search	lookup	buy
1	190	184	780
2	54	20	504
3	15	39	558
4	18	17	671
5	37	17	309
6	5	11	250
7	10	7	300

Search Average response time = 329/7= 47ms.

Lookup Average response time= 289/7= 41ms.

Buy Average response time = 3372/7= 481ms

2-To maintain the caches consistency, any buy request will remove update and search data from cache for the book in buy request, after buy request if user sends lookup request for this item he/she will not find it in the cache because it is removed from it due to previous buy request, in this case the caching of lookup request for this book will not be very useful especially if this book is very trendy and a lot of users send buying requests in small period of time.